***2023***



**逻辑与计算机系统设计 ·实验报告·**

j0242087[1]

|  |  |
| --- | --- |
| 专 业： | 数据科学与大数据技术 |
| 班 级： | 大数据2101班 |
| 学 号： | U202115652 |
| 姓 名： | 李嘉鹏 |
| 电 话： | 18986139698 |
| 邮 件： | [u202115652](mailto:1633741074@qq.com)@hust.edu.cn |
| 完成日期： | 2023-06-26 |



# 运动码表系统设计

## 设计要求

利用logisim平台中现有运算部件构建一个运动码表系统，主要包括7段数码管驱动电路、2选1选择器（1位或16位）、无符号比较器设计（4位或16位）、并行加载寄存器（4位或16位）、BCD计数器状态机及输出函数、BCD计数器（1位十进制）、码表计数器（4位十进制）、码表显示驱动、码表控制器状态机及输出函数、码表控制器、码表数据通路的设计。系统集成后，可支持开始计时、停止计时、存储时间、复位、保存或显示最短时间等功能。

## 方案设计

### 7段数码管驱动电路设计

在logisim中打开本节实验包中的StopWatch.circ文件，本实验所有设计均在这个文件里完成。首先观察数码管，共有7段（这里不考虑小数点），如图1-1所示。这7段中每一段都由一个独立信号控制，信号为1时代表该管亮起，为0时代表不亮起。

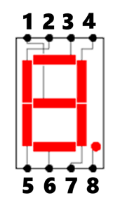


图1-1：7段数码管结构示意图

电路引脚及功能如表1-1所示。

表1-1：7段数码管电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| X3~X0 | 输入 | 1 | 4位BCD码输入 |
| Seg1~Seg7 | 输出 | 1 | 7位数码管驱动信号 |

因此，在设计数码管电路时，首先根据4位BCD码（8421 BCD码）计算出对应输入十进制下的值。设BCD码从高位到低位分别为X3、X2、X1、X0，则计算的方式是：

接下来找出这个值在7段数码管显示下具体亮起的数码管，对这些亮起的数码管，信号赋为1，其余的数码管信号赋为0即可。具体实验中，通过Project→Analyze Circuit（分析组合逻辑电路）菜单下的Table功能，补全7段数码管驱动信号的真值表即可，如图1-2所示。



图1-2：7段数码管电路真值表

随后，点击“Build Circuit”自动生成电路即可实现预期的功能。Logisim中的电路图如图1-3所示。

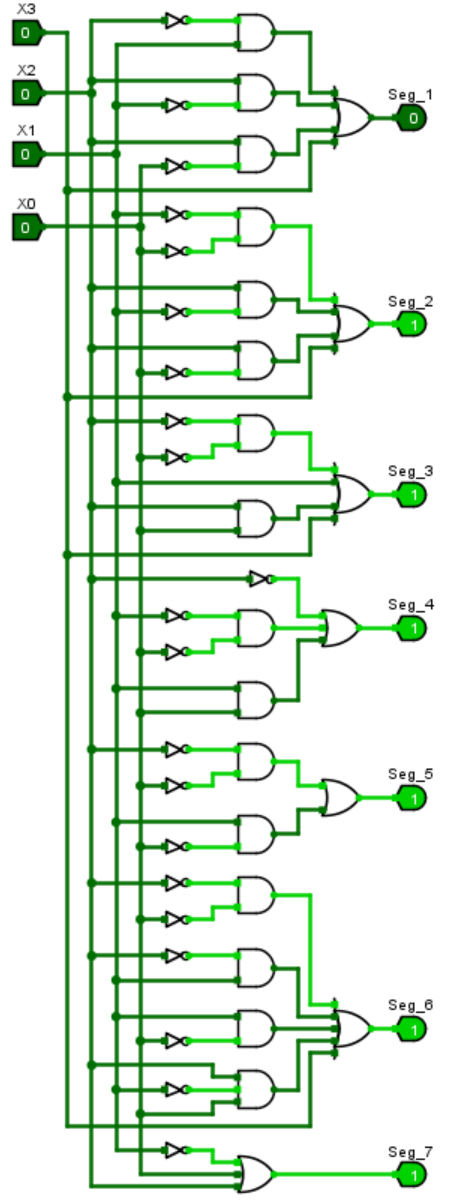


图1-3：7段数码管电路图

### 2选1选择器设计（1位）

本关要利用基本逻辑门构成1位的2路选择器，电路引脚及功能如表1-2所示。

表1-2：2选1选择器（1位）电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| X1~X0 | 输入 | 1 | 2路输入的第0路和第1路 |
| Sel | 输入 | 1 | 选择控制端 |
| Out | 输出 | 1 | 若Sel=0，则选择X0，否则选择X1 |

由于输入的数据只有1位，这里只涉及到基本的逻辑表达式，即

根据上面的表达式，很容易能画出电路结构图。电路图如图1-4所示。

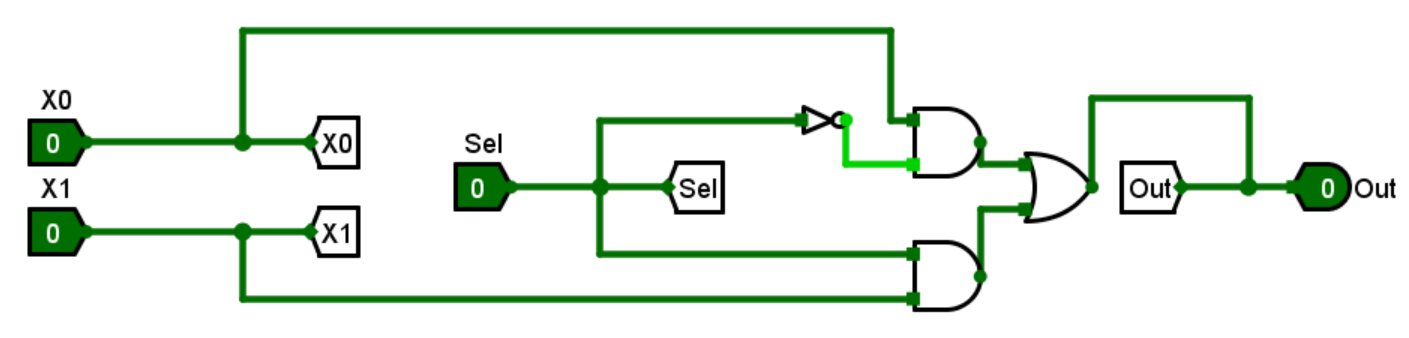


图1-4：2选1选择器（1位）电路图

### 2选1选择器设计（16位）

本关要在上一关的1位2路选择器的基础上，构建16位的2路选择器。很容易想到的思路是把输入的16位数据逐个拆开，拆成16个1位的数据，并使用16个1位2路选择器进行选择即可。这里，电路的引脚和功能和表1-2中的基本一致，唯一的区别在于X0和X1都是16位的。

利用分线器（Splitter）分离开X0和X1的16位数据，并分别置入16个1位2路选择器的输入端，将结果按位输出到Out即可。电路图如图1-5所示。



图1-5：2选1选择器（16位）电路图

### 无符号比较器设计（4位）

本关要设计实现四位无符号数的比较器。电路引脚及功能如表1-3所示。

表1-3：无符号比较器（4位）电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| X3~X0 | 输入 | 1 | 4位输入X |
| Y3~Y0 | 输入 | 1 | 4位输入Y |
| Great | 输出 | 1 | X大于Y |
| Equal | 输出 | 1 | X等于Y |
| Less | 输出 | 1 | X小于Y |

现在要比较X和Y的值，并根据X>Y、X=Y、X<Y这三种情况给出对应的Great、Equal和Less输出。这一电路有8个输入，真值表表项共有项，如果只是列出全部的情况则过于繁琐。实际上，可以利用逻辑表达式的方式简化这一问题。

若X=Y，则显然有成立，即输入X和Y在每一位上都相等。

若X>Y，则满足，或，或，或。

若X<Y，也就是Y>X，则满足，或，或，或。

据此写出Great、Equal和Less的逻辑表达式（两个变量的与运算用空格表示，或运算用“+”表示，非运算用“~”表示，后文中的复杂逻辑表达式均用该方式表示）：

然后自动生成电路即可。电路图如图1-6所示。

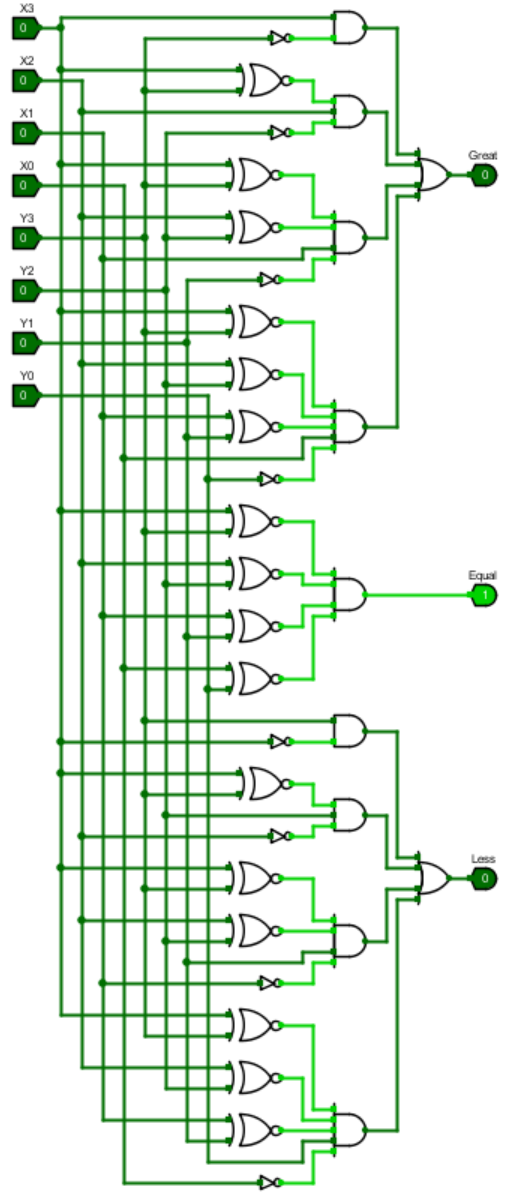


图1-6：无符号比较器（4位）电路图

### 无符号比较器设计（16位）

本关要利用上面设计好的4位无符号比较器构建16位无符号比较器。这里，电路的引脚和功能和表1-3中的基本一致，唯一的区别在于X和Y都是16位的。

在实验中，可以把16位的输入分成四部分，每部分包含4位，并分别输入到四个4位无符号比较器中。然后，综合这四个4位无符号比较器输出的结果，得到整体的比较结果。

若X=Y，则四个比较器的结果显然都为Equal，即输入X和Y在每一位上都相等。

若X>Y，则满足最高位比较器的结果为Great，或最高位比较器的结果为Equal且次高位比较器的结果为Great，或最高位比较器的结果为Equal且次高位比较器的结果为Equal且次次高位比较器的结果为Great，或高位比较器的结果为Equal且次高位比较器的结果为Equal且次次高位比较器的结果为Equal且最低位比较器的结果为Great。

若X<Y，也就是Y>X，与上面的分析类似。

根据上述分析即可连接电路。电路图如图1-7所示。

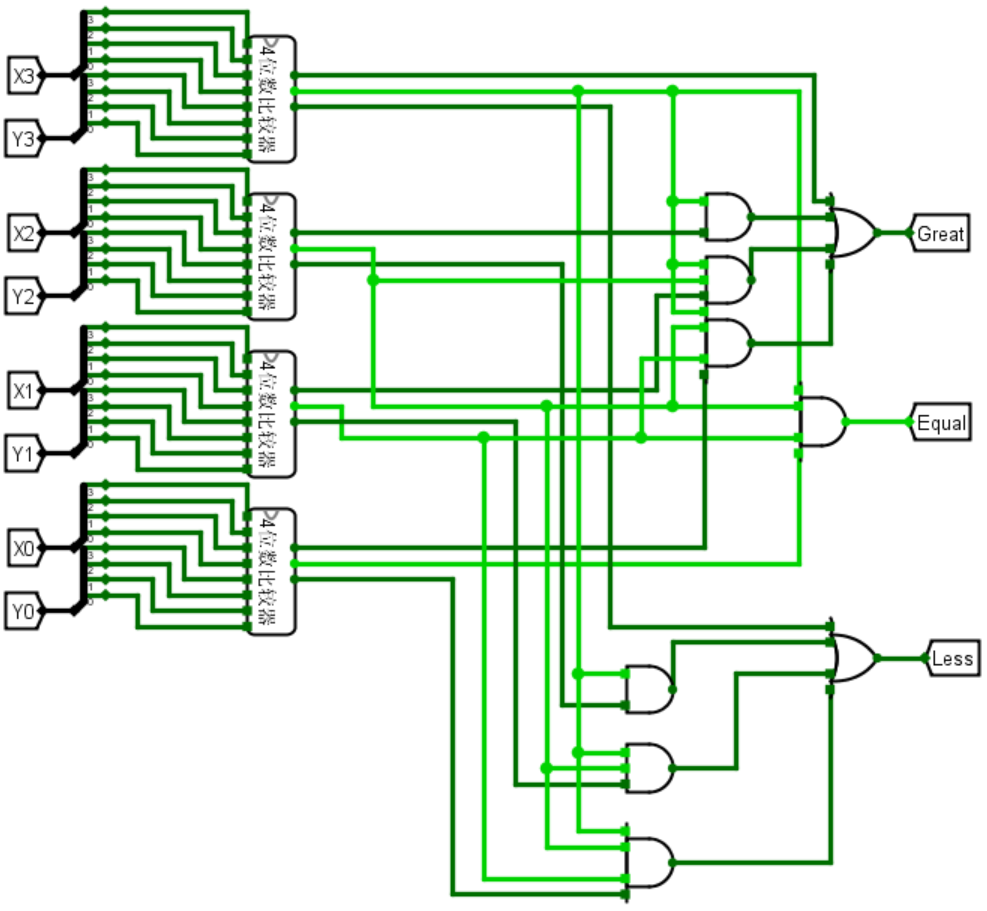


图1-7：无符号比较器（16位）电路图

### 并行加载寄存器设计（4位）

下面要利用D触发器构成4位并行加载寄存器，要求上升沿触发，具有高电平使能端。寄存器是由具有存储功能的触发器组合而成的，其中一个触发器存储一位二进制代码。因此，存放4位二进制代码的寄存器，需要4个D触发器构成。电路引脚及功能如表1-4所示。

表1-4：并行加载寄存器（4位）电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| Clock | 输入 | 1 | 时钟脉冲，上升沿有效 |
| Din | 输入 | 4 | 寄存数据输入端 |
| En | 输入 | 1 | 使能输入端，高电平有效 |
| Q | 输出 | 4 | 寄存数据输出端，En=1且Clock上升沿时Q=Din |

D触发器的特点是其次态和现态一致，因此只需要把输入用分线器分开，分别输入4个D触发器的“D”输入端，并将这4个D触发器的次态输出按位输出到Q即可。注意D触发器需要接入时钟Clock和使能端En，否则无法正常工作。电路图如图1-8所示。



图1-8：并行加载寄存器（4位）电路图

### 并行加载寄存器设计（16位）

本关要利用上一关设计好的4位并行加载寄存器构建16位并行加载寄存器，要求沿上升沿触发，且具有高电平使能端。这里，电路的引脚和功能和表1-4中的基本一致，唯一的区别在于输入Din和输出Q都是16位的。

分析封装好的4位寄存器，发现其输入有时钟、4位输入数据和使能端。因此，在处理16位并行加载寄存器时，可以使用4个封装好的4位寄存器，按要求对Q和Din进行分线即可。电路图如图1-9所示。

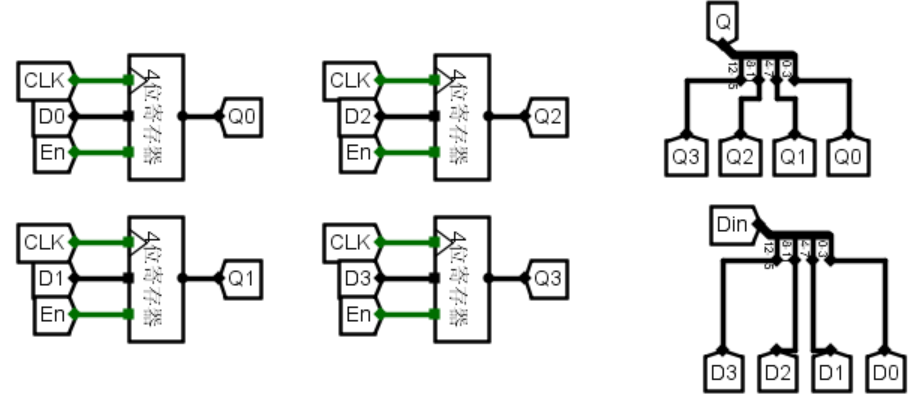


图1-9：并行加载寄存器（16位）电路图

### BCD计数器状态机设计

本关要设计8421 BCD码（十进制）计数器的状态机。电路引脚及功能如表1-5所示。

表1-5：BCD计数器状态机电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| S3~S0 | 输入 | 4 | 当前状态S |
| N3~N0 | 输入 | 4 | 次态输出N，N=S+1 |

根据题目要求，在“同步时序电路状态转换表.xlsx”中完成状态转换真值表。由于是计数器，因此现态为0时次态为1，现态为1时次态为2，以此类推，现态为时次态为，现态为9时次态为0（代表进位了）。由此即可画出状态转换真值表，如图1-10所示。注意这里没有输入信号，不需要填写表格中间的部分。



图1-10：BCD计数器状态转换真值表

填写完成后，在Excel表格的“状态转换函数自动生成”下自动生成次态输出逻辑表达式。其中，，，，。然后在Logisim中自动生成电路即可，电路图如图1-11所示。

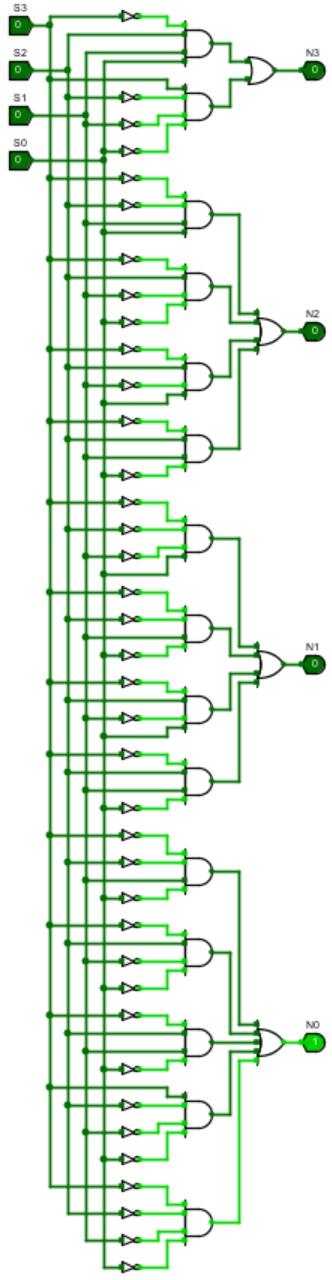


图1-11：BCD计数器状态机电路图

### BCD计数器输出函数设计

本关要设计BCD计数器的输出函数组合逻辑，生成计数器的进位输出信号，该输出信号仅与状态信号有关（即Moore型输出）。电路引脚及功能如表1-6所示。

表1-6：BCD计数器输出函数电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| S3~S0 | 输入 | 4 | 当前状态S |
| Cout | 输出 | 1 | 进位输出，S=9时，Cout=1 |

具体来说，输出当且仅当当前状态S=9时（即S3=S0=1，S2=S1=0）为1，其余时刻均为0。因此可以直接手动绘制电路。电路图如图1-12所示。

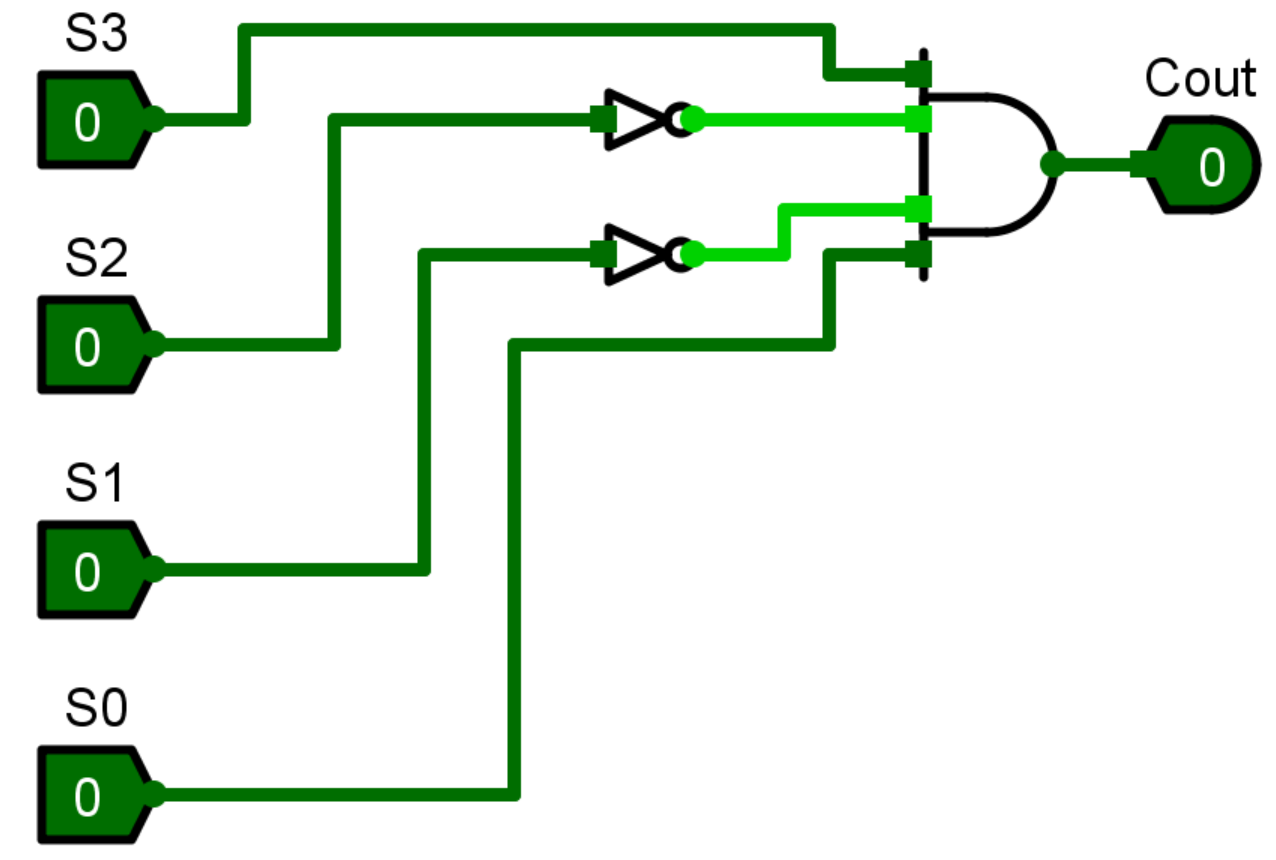


图1-12：BCD计数器输出函数电路图

### BCD计数器设计（1位十进制）

下面要利用已经设计好的BCD计数器的状态机、输出函数，采用D触发器构建最终的BCD计数器。电路引脚及功能如表1-7所示。

表1-7：BCD计数器（1位十进制）电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| CLK | 输入 | 1 | 时钟输入 |
| Rst | 输入 | 1 | 异步复位信号，为1时Q=0 |
| En | 输入 | 1 | 使能信号，为1时进行计数，Q=Q+1 |
| Q | 输出 | 4 | 计数器计数输出 |
| Cout | 输出 | 1 | 进位输出，计数到9时输出为1 |

首先利用分线器把Q分为4位，并将Q3~Q0的每一位都输入到状态机中，得到状态转换后的次态QN3~QN0。然后将次态的4位数分别输入4个D触发器，使其输出覆盖现态的Q3~Q0。最后，同样还要将新的“现态”Q3~Q0输入到封装好的输出函数模块，从而得到进位输出。电路图如图1-13所示。

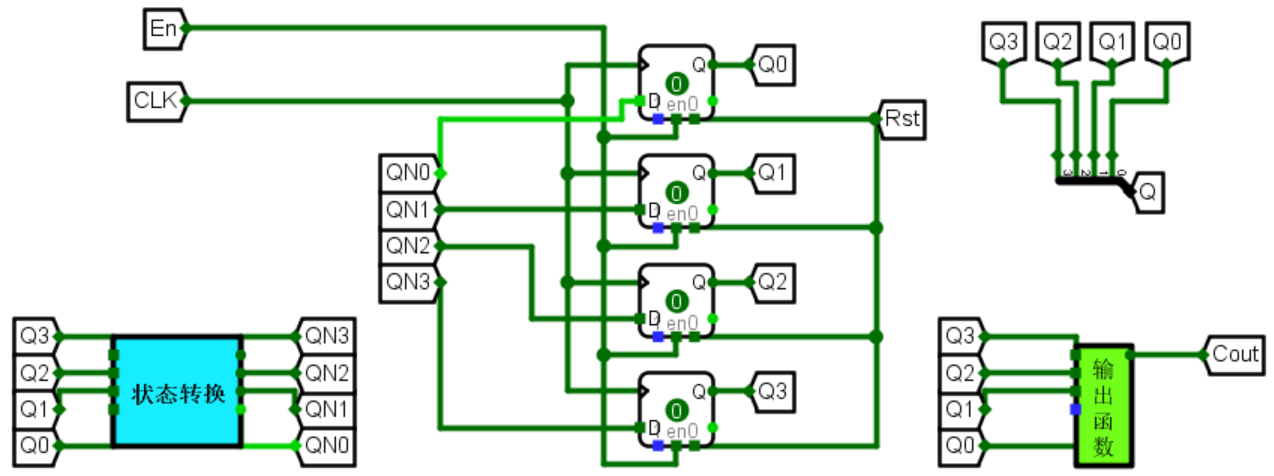


图1-13：BCD计数器（1位十进制）电路图

### 码表计数器设计（4位十进制）

本关要利用上一关已经设计好的BCD计数器，级联构建4位十进制计数器。由于码表的计时范围为00.00~99.99，因此需要4位十进制计数器。这里，电路的引脚和功能和表1-7中的基本一致，唯一的区别在于没有进位输出Cout了。

设计过程中，需要考虑低位向高位进位的影响。第个十进制计数器开始工作需要满足前个十进制计数器的进位输出均为1，因此需要用与门连接到第2~4个计数器的使能En端。电路图如图1-14所示。

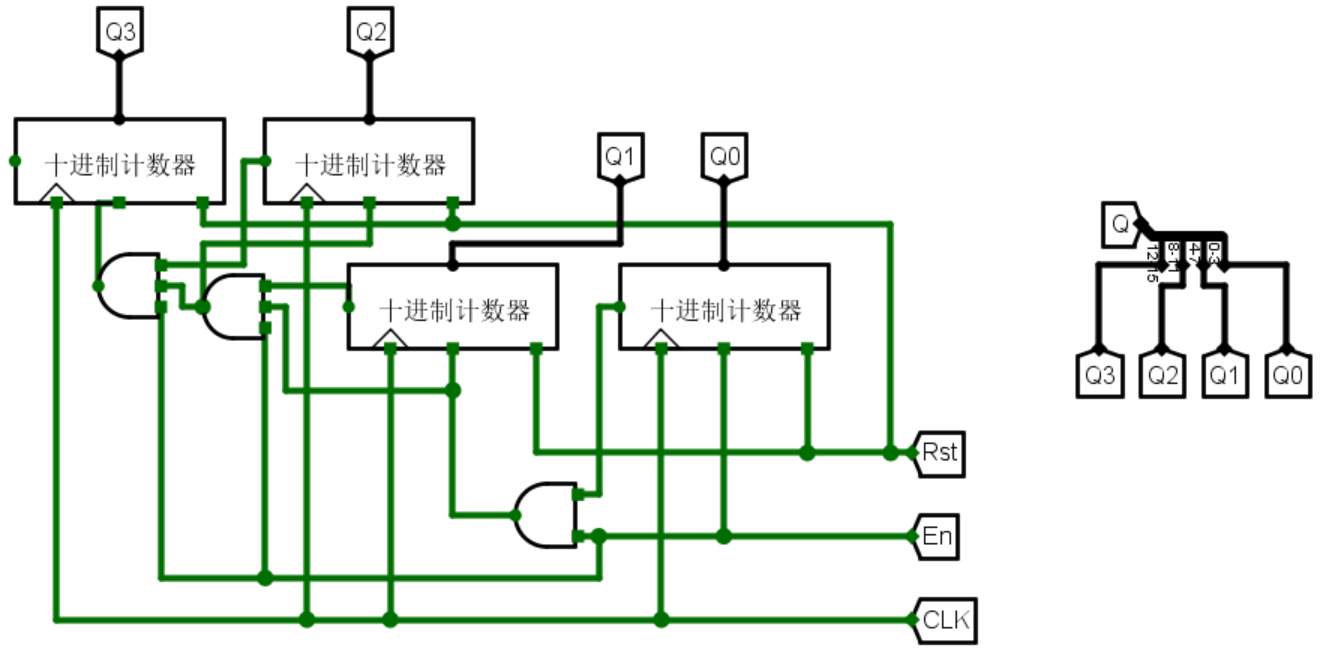


图1-14：码表计数器（4位十进制）电路图

### 码表显示驱动设计

本关要利用第1关的7段数码管驱动电路级联构建4位十进制显示驱动电路。电路引脚及功能如表1-8所示。

表1-8：码表显示驱动电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| Din | 输入 | 16 | 4位十进制（16位BCD码） |
| DispInfo | 输出 | 32 | 4个8段（7段+小数点）数码驱动信号 |

首先用分线器把Din分为四个部分，每部分包含4位数字。然后分别把这四部分输入到封装好的数码管驱动模块中。注意到码表的显示格式为xx.xx，也就是第二个数码管驱动电路的第8段数码驱动信号为1，从而实现显示小数点的功能，而其余三个数码管驱动电路的第8段数码驱动信号均为0。电路图如图1-15所示。

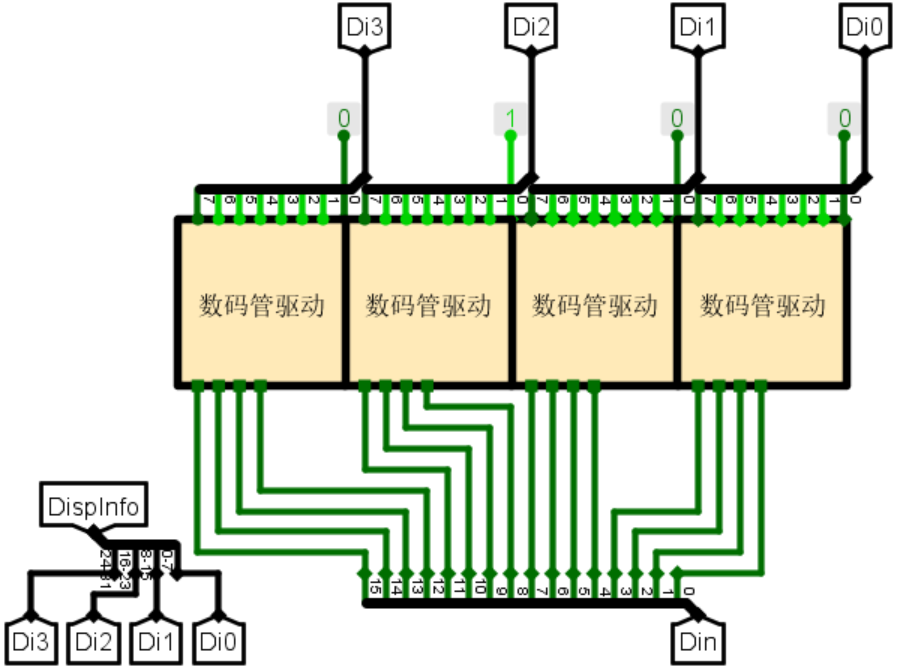


图1-15：码表显示驱动电路图

### 码表控制器状态机设计

本关要设计码表控制器的状态机（即状态转换组合逻辑）。

电路引脚及功能如表1-9所示。

表1-9：码表控制器状态机电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| start | 输入 | 1 | 开始计时信号 |
| stop | 输入 | 1 | 停止计时信号 |
| store | 输入 | 1 | 存储计时记录信号 |
| reset | 输入 | 1 | 计时复位信号，记录恢复为99.99 |
| newrecord | 输入 | 1 | 新记录信号 |
| S2~S0 | 输入 | 3 | 现态S |
| N2~N0 | 输出 | 3 | 次态N |

首先分析码表可能存在的状态，主要有复位、清零、计时、计时停止、存储、查看等。根据码表功能，可以绘制出码表的状态转移图，如图1-16所示。

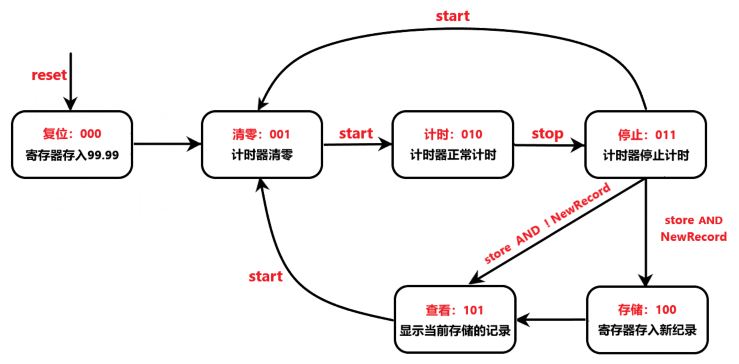


图1-16：码表控制器状态机状态转移图

根据这一状态转移图，可以在“同步时序电路状态转换表.xlsx”中完成状态转换真值表，如图1-17所示。

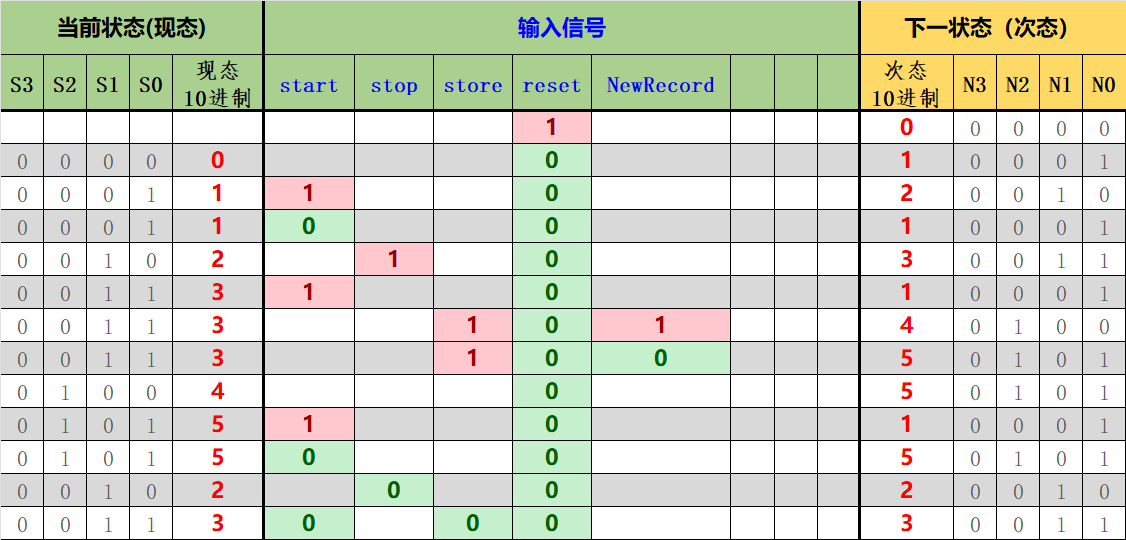


图1-17：码表控制器状态转换真值表

需要注意的是，状态在转移过程中，除了图中显式标明的状态转移路径，还存在一些隐含的状态转移条件和状态维持条件。具体列举如下：

（1）只有给出reset指令时，才会转移到状态0（复位），其它任何条件下都不会转移到状态0；

（2）在状态为1（清零）时，若不给出start指令，则会一直维持状态1；

（3）状态为4（存储）时，若不给出任何指令，则会直接跳转到状态5（查看）。

填写完成后，在Excel表格的“状态转换函数自动生成”下自动生成次态输出逻辑表达式。其中，满足下面的逻辑表达式：

然后在Logisim中自动生成电路即可，电路图如图1-18所示。

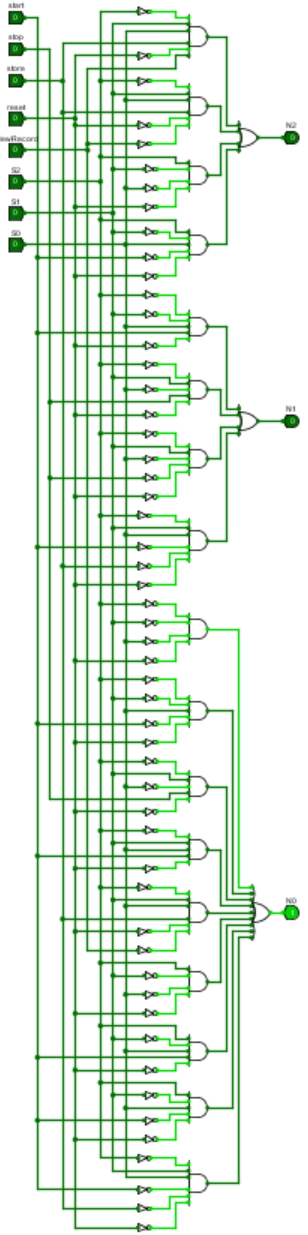


图1-18：码表控制器状态转换电路图

### 码表控制器输出函数设计

本关要设计码表控制器的输出函数组合逻辑（Moore型输出），需要结合运动码表系统数据通路完成码表控制器输出函数真值表。电路引脚及功能如表1-10所示。

表1-10：码表控制器输出函数电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| S2~S0 | 输入 | 3 | 当前状态S（PS） |
| SDsel | 输出 | 1 | 最好成绩记录的选择信号 |
| SDen | 输出 | 1 | 保存最好成绩记录的寄存器的使能信号 |
| DPsel | 输出 | 1 | 显示计时成绩记录的选择信号 |
| TMen | 输出 | 1 | 码表计时器使能信号 |
| TMreset | 输出 | 1 | 码表计时器复位信号 |

根据题目给定的测试用例，可以作出码表控制器的输出函数真值表，如图1-19所示。注意这里没有输入信号，不需要填写表格中间的部分。

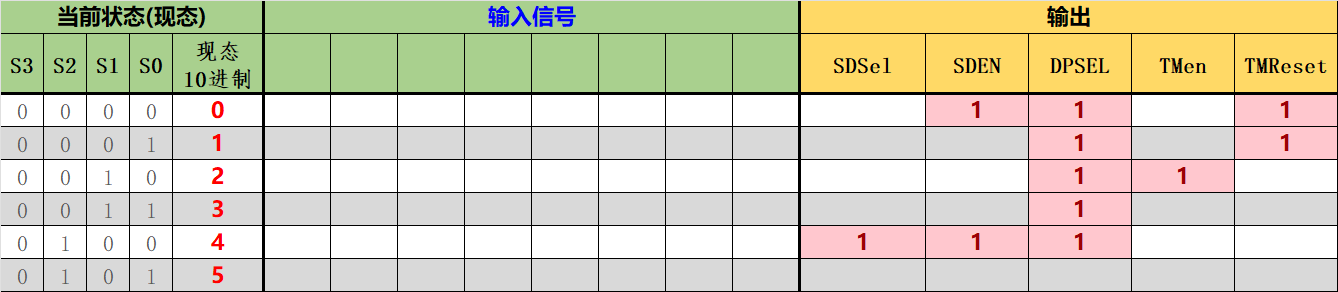


图1-19：码表控制器输出函数真值表

填写完成后，在Excel表格的“输出函数自动生成”下自动生成输出函数的逻辑表达式。其中，5个不同的输出分别满足以下逻辑表达式：

，，，

，

然后在Logisim中自动生成电路即可，电路图如图1-20所示。

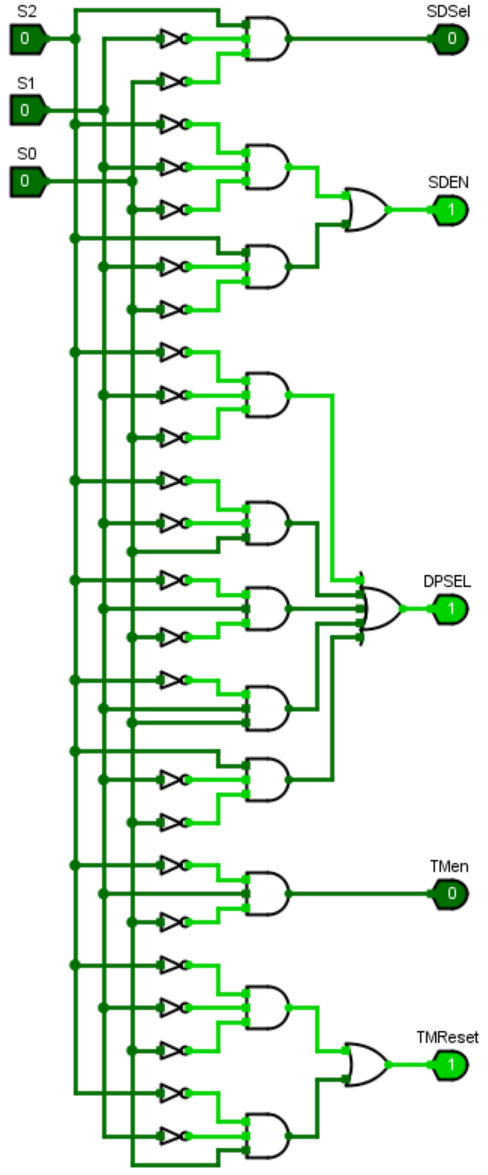


图1-20：码表控制器输出函数电路图

### 码表控制器设计

跟据同步时序逻辑电路基本模型，利用已经设计好的码表控制器的状态机（状态转换组合逻辑）、输出函数，采用D触发器（或寄存器）构建最终的码表控制器，完成有限状态机的组装。

至此，基本已经可以整合码表的全部功能，包括开始计时、停止计时、存储计时记录、计时复位、选择最好成绩记录、保存最好成绩记录、显示计时记录、计时器复位等。电路引脚及功能如表1-11所示。

表1-11：码表控制器电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| CLK | 输入 | 1 | 时钟脉冲 |
| start | 输入 | 1 | 开始计时信号 |
| stop | 输入 | 1 | 停止计时信号 |
| store | 输入 | 1 | 存储计时记录信号 |
| reset | 输入 | 1 | 计时复位信号，记录恢复为99.99 |
| newrecord | 输入 | 1 | 新的最好成绩记录信号 |
| SDsel | 输出 | 1 | 最好成绩记录的选择信号 |
| SDen | 输出 | 1 | 保存最好成绩记录的寄存器的使能信号 |
| DPsel | 输出 | 1 | 显示计时成绩记录的选择信号 |
| TMen | 输出 | 1 | 码表计时器使能信号 |
| TMreset | 输出 | 1 | 码表计时器复位信号 |

这一关是对前面的模块的级联和汇总，按照封装后的状态转换模块和输出函数模块完成对应的连线即可。

在状态转换过程中，设S2~S0为现态，N2~N0为次态，这六个引脚都是中间变量。电路图如图1-21所示。

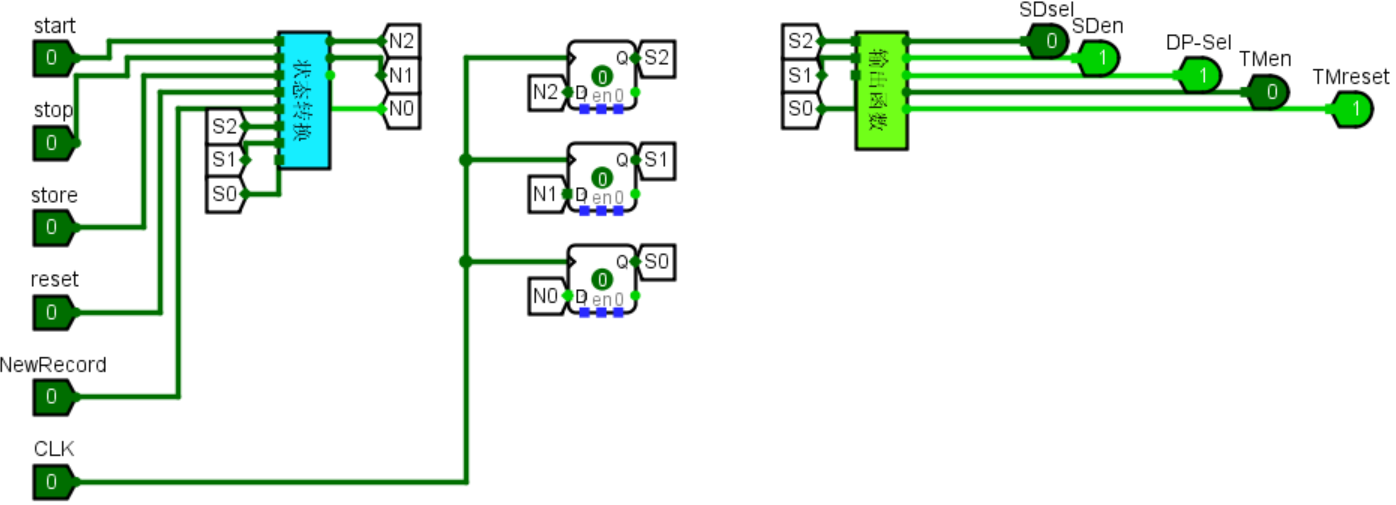


图1-21：码表控制器电路图

### 运动码表数据通路设计（系统集成）

这是整个运动码表实验的最后一步，需要利用前面15关中设计好的选择器、比较器、寄存器、计数器、码表控制器等部件构建运动码表系统数据通路。注意初始记录99.99和最好成绩记录必须送入选择器0号端口。电路引脚及功能如表1-12所示。

表1-12：运动码表数据通路电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| CLK | 输入 | 1 | 时钟脉冲 |
| TestMode | 输入 | 1 | 在线评测开关，0为本地模式，1为在线模式 |
| start | 输入 | 1 | 开始计时信号 |
| stop | 输入 | 1 | 停止计时信号 |
| store | 输入 | 1 | 存储计时记录信号 |
| reset | 输入 | 1 | 计时复位信号，记录恢复为99.99 |
| Time | 输出 | 16 | 计时成绩或成绩记录 |

首先把码表控制器的输入输出信号都按要求进行连接。具体有以下几项：

（1）将输入Start1、Stop1、Store1、Reset1、CLK、NewRecord都连接到码表控制器上；

（2）将时钟CLK连接到16位并行数据寄存器SD和计时器TM上；

（3）将码表控制器的输出TM-EN和TM-Reset连接到计时器TM上，确保计时器的正常运行和复位功能；

（4）将码表控制器的输出DP-SEL连接到码表显示信号的选择器上，确定显示的计时记录来源；

（5）将码表控制器的输出SD-EN连接到16位寄存器上，确定寄存器是否保存最好成绩记录；

（6）将码表控制器的输出SD-SEL连接到最好成绩记录的选择器上，确定是否选择最短计时记录。

接着要构建数据通路。计时器的输出需要连接到最好成绩记录的选择器和码表显示信号的选择器上，从而为各类功能提供当前的计时记录。由16位寄存器输出的数据需要送给最好成绩记录以及比较器，并与当前的计时记录进行比较，以输出新的最好成绩记录信号NewRecord。

综合以上全部分析，可以连接出电路图如图1-22所示。

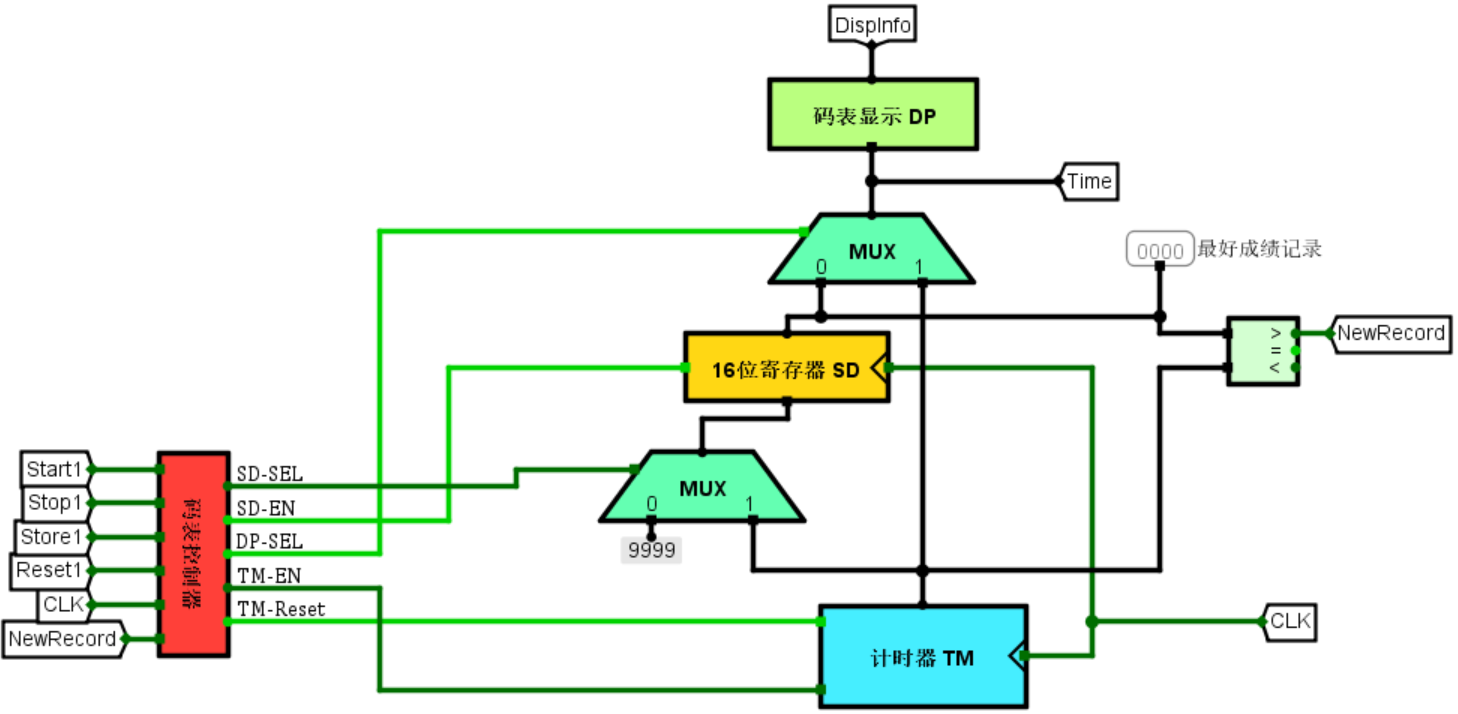


图1-22：运动码表数据通路（系统集成）电路图

## 实验步骤

**（1）基本电路模块构建与测试**

根据1.2中的设计方案构建各个电路部件，包括数码管驱动电路、选择器、比较器、寄存器、BCD计数器、码表计数器、码表显示驱动、码表控制器。

电路构建完成后，在logisim的测试电路下运行测试。首先运行“2路选择器自动测试”电路，成功通过，显示“PASS”字样，如图1-23所示。

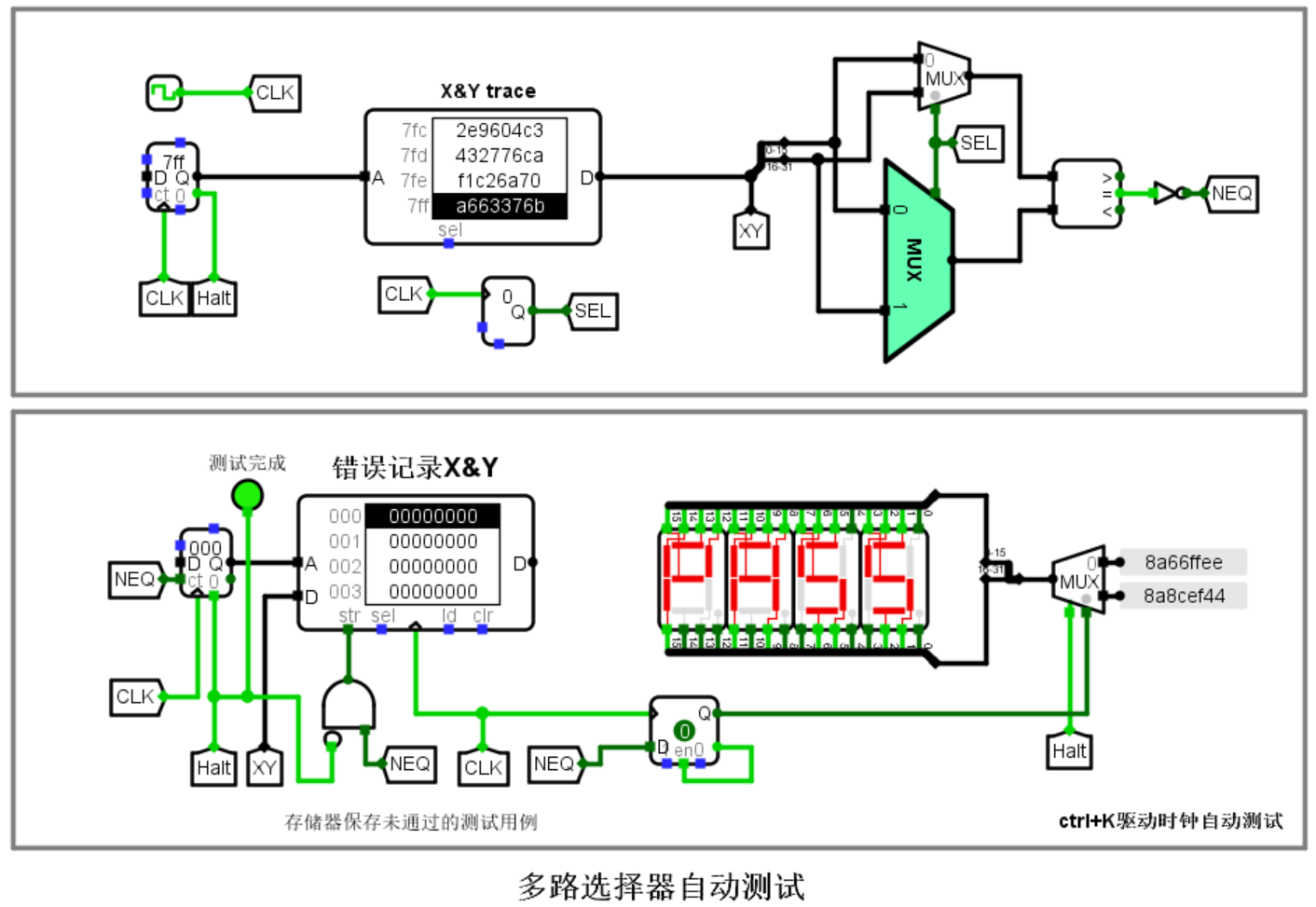


图1-23：2路选择器自动测试

然后运行“16位无符号比较器自动测试”电路，成功通过，显示“PASS”字样，如图1-24所示。



图1-24：16位无符号比较器自动测试

接着运行“码表计数器自动测试”电路，将时钟频率设为100Hz，按Ctrl+K键启动码表，计时功能正常，如图1-25所示。



图1-25：码表计数器自动测试

**（2）运动码表功能测试**

对所有模块整合后，打开“★运动码表”电路，构建测试集并手动运行测试。

首先按Start按钮，码表开始正常计时，如图1-26所示。

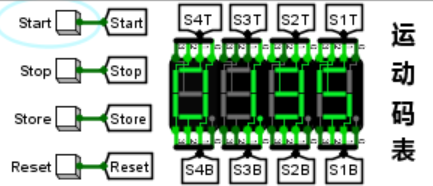


图1-26：运动码表Start功能测试（开始计时）

接下来按Stop按钮，可以看到码表的计时记录维持不变，为停止状态，显示为“02.86”。按Store按钮保存本次记录为最好时间记录，可以看到数据通路电路中“最好成绩记录”输出由9999变为本次时间记录286（单位：毫秒），如图1-27所示。

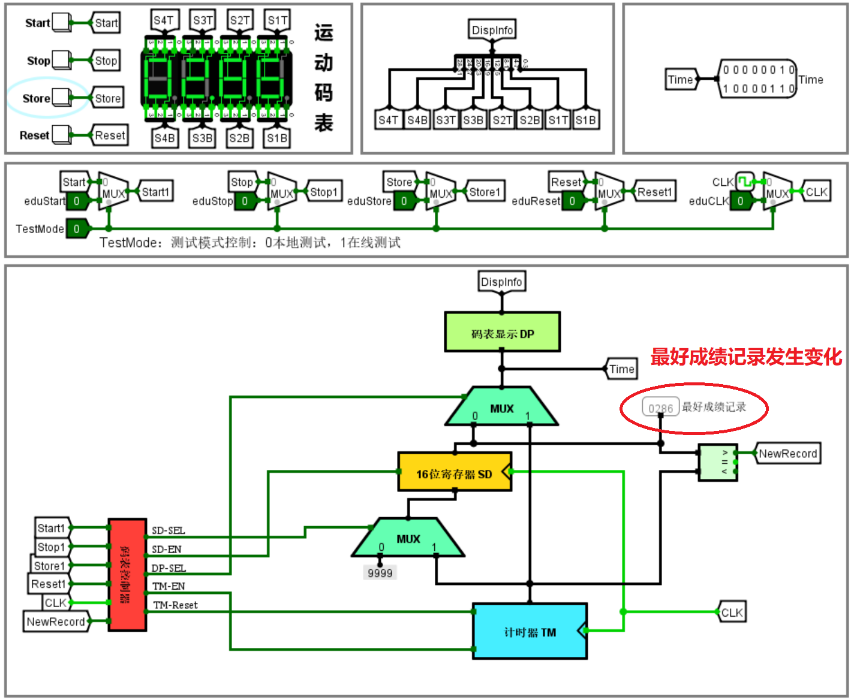


图1-27：运动码表Stop、Store功能测试（停止计时和存储）

最后按Reset按钮，码表复位，显示为“00.00”，最好成绩记录变为9999，如图1-28、图1-29所示。

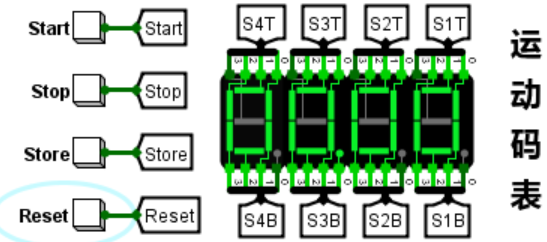


图1-28：运动码表Reset功能测试（复位后码表显示）

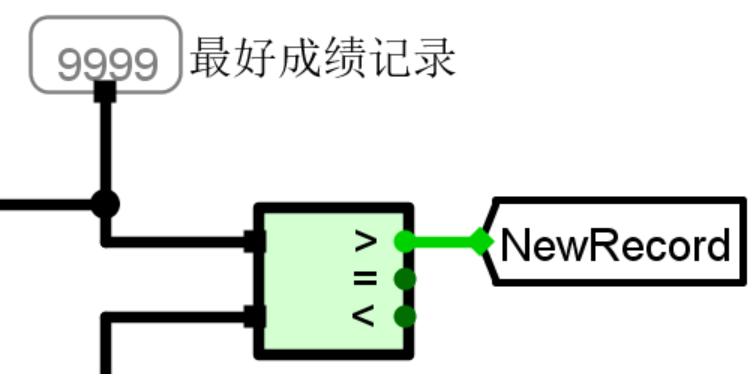


图1-29：运动码表Reset功能测试（复位后最好成绩记录变为9999）

## 故障与调试

### 引脚悬空时的信号传输问题

**故障现象：**在复杂电路中因为连线问题导致某些引脚没有接上（悬空），导致对应的引脚在输出结果时输出的是“xxxx”，如图1-30所示。

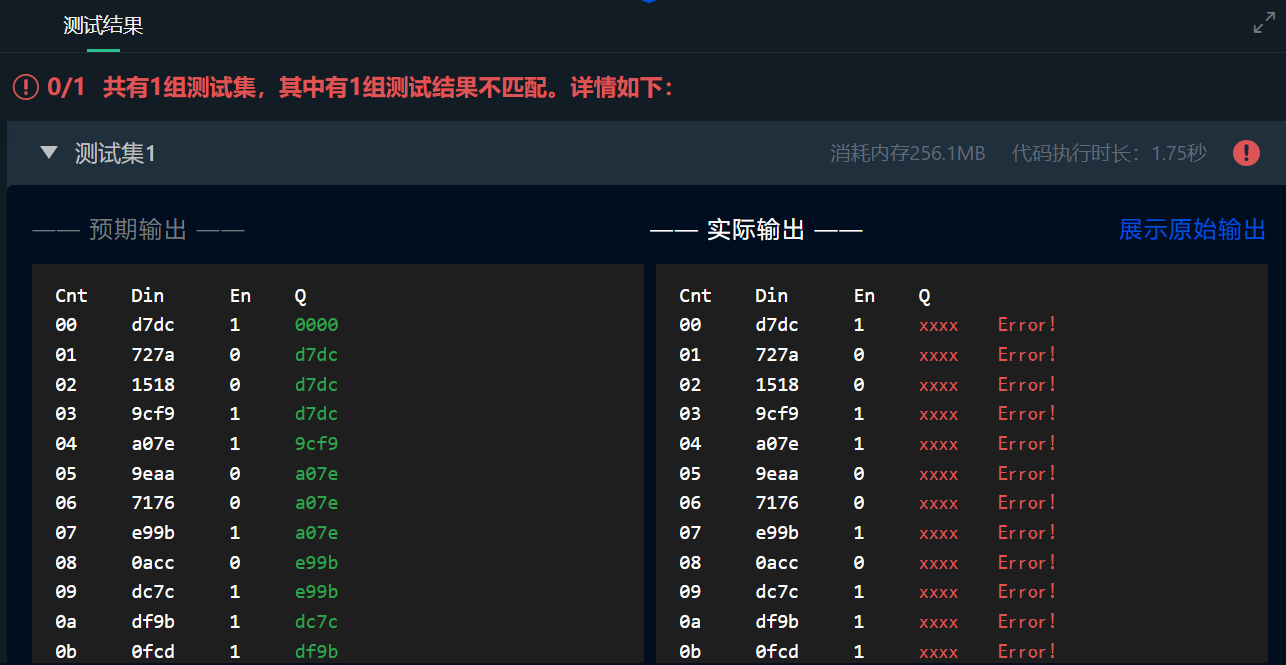


图1-30：引脚悬空时信号无法正常传输

**原因分析：**如图1-31所示，在处理输入Q时利用分线器将其分为4位，分别为Q1~Q4，但在输出结果时没有把Q1~Q4按位送给Q，导致Q引脚悬空，无法读取每一位的值。这是由于在分线器分开后没有再次汇合，属于设计并行寄存器时的失误。

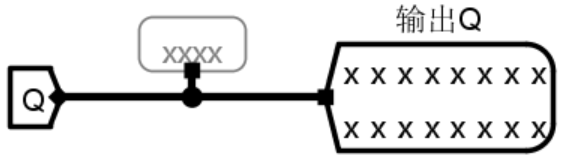


图1-31：引脚悬空

**解决方案：**再引入一个分线器，把Q1~Q4按位送给Q即可解决引脚悬空的问题。

### 对时钟脉冲信号非法处理的问题

**故障现象：**在设计4位BCD计数器时，刚开始随着时钟脉冲的产生，评测结果正确，但过了一段时间后，后面的输出Q就开始出现错误结果，如图1-32所示。



图1-32：对时钟脉冲信号非法处理后的评测结果

**原因分析：**如图1-33所示，在处理每个D触发器的使能端En时，简单地认为当且仅当En为高电平时才需要把时钟脉冲信号CLK输入每个D触发器的时钟端口，否则只需往时钟端口输入0即可。后来经过老师指导后发现不允许对时钟信号做任何操作，只能直接连接到触发器的时钟端口上。

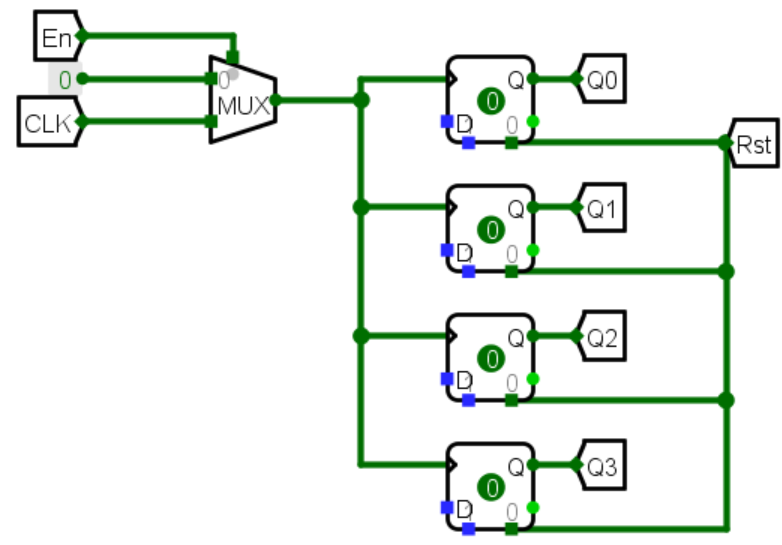


图1-33：对时钟脉冲信号的非法处理（见左上角）

**解决方案：**只需要对D触发器增加一个单独的使能端En，并将时钟和使能输入分别连接4个D触发器相应的输入端即可。

## 测试与分析

运动码表的测试与分析已经在上面1.3节完成，此处不再赘述。最终在头歌在线平台提交测评，成功通过了16关，获得满分。

## 实验总结

1. 本次实验中，我利用了logisim平台中现有运算部件构建一个运动码表系统。总体方案上，首先设计了局部的模块，包括7段数码管驱动电路、2选1选择器（1位或16位）、无符号比较器设计（4位或16位）、并行加载寄存器（4位或16位）、BCD计数器状态机及输出函数、BCD计数器（1位十进制）、码表计数器（4位十进制）、码表显示驱动、码表控制器状态机及输出函数、码表控制器、码表数据通路的设计。最后，对上述模块进行集成。
2. 本次实验的运动码表系统最终实现了开始计时、停止计时、存储时间、复位、保存或显示最短时间等功能。

## 实验心得

在本次实验中，我通过对码表各个基本模块的构建以及对运动码表数据通路的设计，最终完成了一个完整的运动码表系统。在这个过程中，我深刻理解了数字电路设计的基本原理和方法，包括时序逻辑、组合逻辑、状态机设计、时钟信号的使用等。通过实践，我更加熟悉了各个基本模块的功能和设计方法，如寄存器的并行加载、计数器的状态转换、数码管的驱动电路等。同时，我也学习到了如何将这些基本模块有机地结合起来，构建出一个完整的系统。在这个过程中，我不断调试和优化电路，解决各种问题和故障，提高了自己的实践能力和解决问题的能力。我主要遇到的问题是引脚悬空和对时钟脉冲的非法处理两种情形，不过经过仔细检查后都得以解决。通过这次实验，我不仅掌握了数字逻辑电路设计的基本技能，还培养了自己的创新意识和实践能力，为今后的学习打下了坚实的基础。

# CPU设计实验

## 设计要求

利用logisim平台中现有运算部件构建一个32位MIPS CPU处理器，包括单周期硬布线CPU、多周期微程序CPU以及多周期硬布线CPU。

该MIPS CPU可支持表2-1中列出的所有指令。具体指令功能与指令格式可以参见群里给出的MIPS基准指令集手册，最终设计好的CPU应该能正常运行给定的冒泡排序测试程序sort.hex，并能在内存中观察到正确结果（即在地址为80处数据降序排序完成）。

表2-1：MIPS CPU支持的指令

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **MIPS指令** | **功能描述** | **备注** |
| 1 | add $rd, $rs, $rt | R[$rd]←R[$rs] + R[$rt] | 溢出产生异常，且不修改R[$rd] |
| 2 | slt $rd, $rs, $rt | R[$rd]←R[$rs] < R[$rt] | 小于置1（有符号比较） |
| 3 | addi $rt, $rs, imm | R[$rt]←R[$rs] + SignExt(imm) | 溢出产生异常 |
| 4 | lw $rt, imm($rs) | R[$rt]←Mem(R[$rs] + SignExt(imm)) | 无 |
| 5 | sw $rt, imm($rs) | Mem(R[$rs] + SignExt(imm))←R[$rt] | 无 |
| 6 | beq $rs, $rt, imm | if(R[$rs] = R[$rt]) PC←PC + SignExt({imm, 00}) | 无 |
| 7 | bne $rs, $rt, imm | if(R[$rs] != R[$rt]) PC←PC + SignExt({imm, 00}) | 无 |
| 8 | syscall | 系统调用，这里用于停机 | 无 |

查阅MIPS基准指令集手册，可以列出上面8条指令的指令码前6位（操作码OP）以及R型指令的后6位（Func），在后续实验设计中需要使用，如表2-2所示。从表中可以看出，只有ADD、SLT、SYSCALL属于R型指令，其OP为0，具体指令类型由Func指定；对于其它指令，具体指令类型由OP指定。同时，第21-25位、16-20位、11-15位分别对应rs、rt、rd寄存器的数据。

表2-2：MIPS CPU支持的指令的指令码OP及Func

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **MIPS指令** | **OP（前6位）** | **Func（后5位）** |
| 1 | add $rd, $rs, $rt | 000000 | 100000 |
| 2 | slt $rd, $rs, $rt | 000000 | 101010 |
| 3 | addi $rt, $rs, imm | 001000 | 非R型指令 |
| 4 | lw $rt, imm($rs) | 100011 | 非R型指令 |
| 5 | sw $rt, imm($rs) | 101011 | 非R型指令 |
| 6 | beq $rs, $rt, imm | 000100 | 非R型指令 |
| 7 | bne $rs, $rt, imm | 000101 | 非R型指令 |
| 8 | syscall | 000000 | 001100 |

## 方案设计

### 单周期MIPS CPU（硬布线）及单周期硬布线控制器设计

利用运算器实验、存储系统实验中构建的运算器、寄存器文件、存储系统等部件以及Logisim中其它功能部件，构建一个32位MIPS CPU单周期处理器。数据通路如图2-1所示。

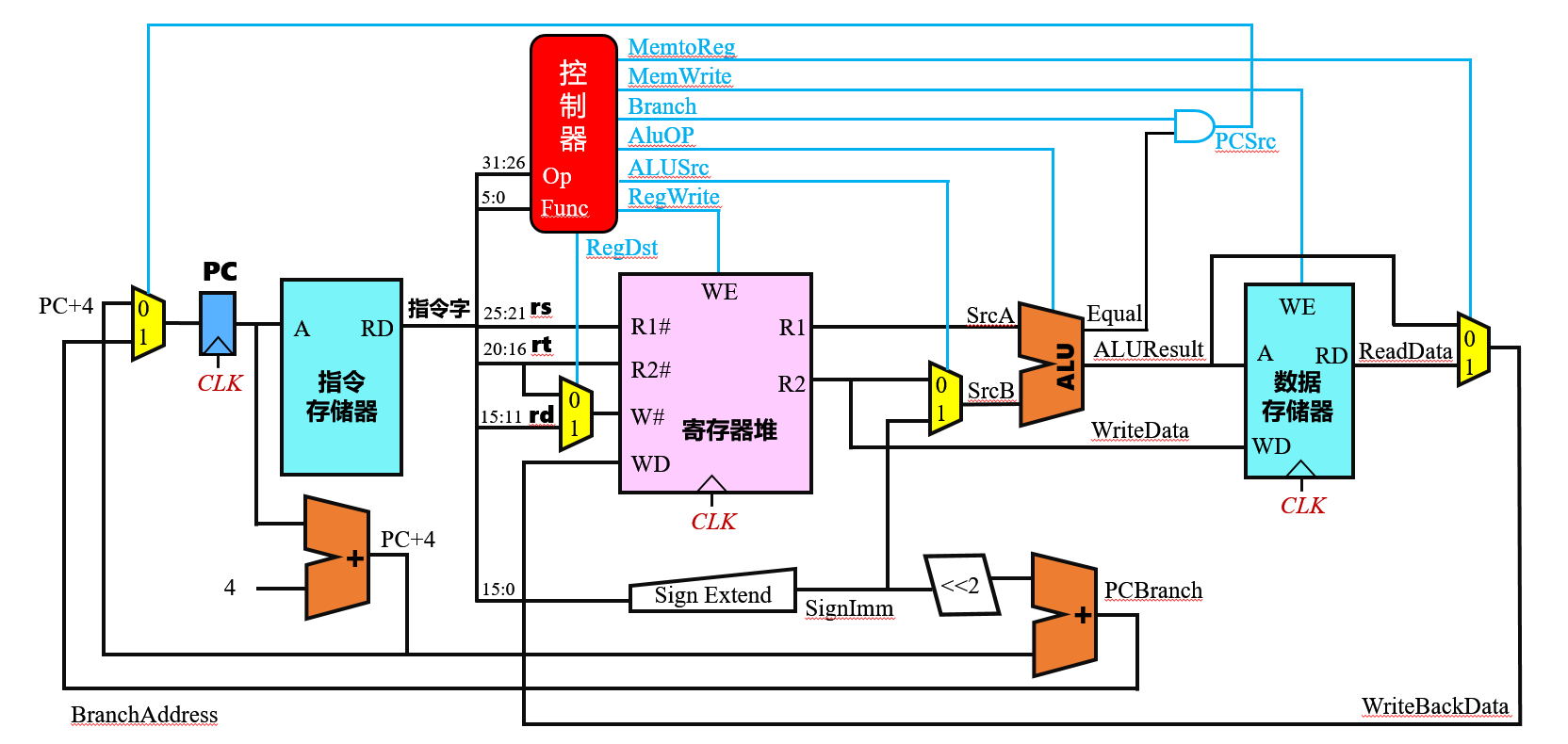


图2-1：单周期MIPS CPU处理器数据通路

电路引脚及功能如表2-3所示。

表2-3：单周期MIPS CPU处理器电路引脚与功能描述

|  |  |  |  |
| --- | --- | --- | --- |
| **引脚** | **输入/输出** | **位宽** | **功能描述** |
| CLK | 输入 | 1 | 时钟信号 |
| PC | 输出 | 32 | 程序寄存器的值 |
| IR | 输出 | 32 | 当前指令字 |
| RegWrite | 输出 | 1 | 寄存器文件写使能控制信号 |
| RDin | 输出 | 32 | 寄存器文件写入端口的数据 |
| MemWrite | 输出 | 1 | 存储器写使能控制信号 |
| MDin | 输出 | 32 | 存储器写入端口的数据 |

整体架构主要包括以下功能部件：

（1）指令计数器PC，用于存放当前指令的地址；

（2）指令存储器IR（使用ROM），存储当前CPU需要执行的指令；

（3）数据存储器DR（使用RAM），存储在指令执行过程中需要操作的数据；

（4）立即数扩展器S-EXT，用于将16位立即数扩展为32位。由于这里的立即数可能是正数或负数，因此需要使用符号扩展功能；

（5）控制器，产生控制信号并控制指令执行的数据通路。这主要是根据指令的OP和Func译码并产生控制信号，由此即可列出各个指令下每个控制信号的值，并得出每个控制信号的逻辑表达式，从而根据指令译码信号实现控制信号的逻辑。这一部分还包含ALU控制器的逻辑，即实现ALU\_OP与OP、Func之间的逻辑关系；

（6）寄存器堆RegFile，提供32个MIPS通用寄存器；

（7）算术逻辑单元ALU，可以提供加或减的运算功能并输出运算结果。

设计好单周期硬布线控制器后，按照图2-1中的数据通路对各个部件进行连接即可。详细流程见实验步骤。

### 多周期MIPS CPU（微程序）及多周期微程序控制器设计

对照多周期MIPS处理器数据通路图，采用微程序控制器的设计方法实现控制器，构造32位的多周期MIPS CPU处理器。数据通路如图2-2所示。

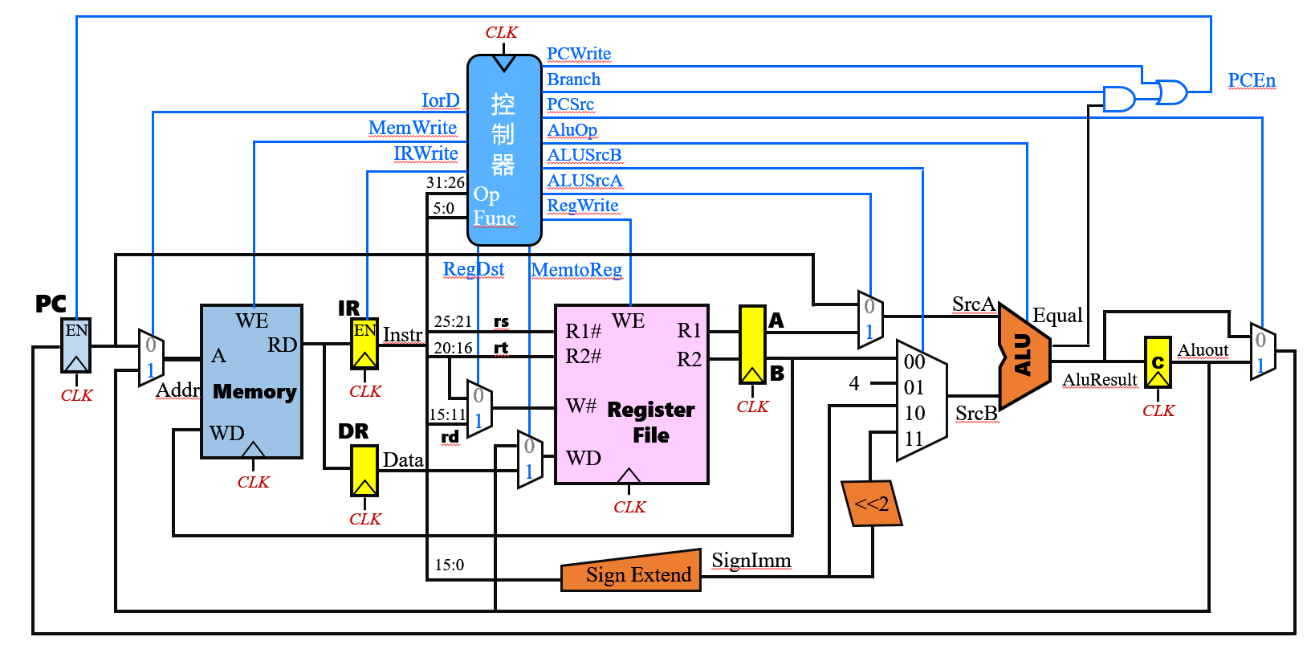


图2-2：多周期MIPS CPU处理器（微程序）数据通路

这里，电路的引脚和功能和表2-3中的完全一致。整体架构主要包括以下功能部件：

（1）指令计数器PC，用于存放当前指令的地址；

（2）存储器Mem（使用RAM），同时相当于指令存储器和数据存储器；

（3）指令寄存器IR，存储当前CPU需要执行的指令；

（4）数据寄存器DR，存储在指令执行过程中需要操作的数据；

（5）立即数扩展器S-EXT，用于将16位立即数扩展为32位，使用符号扩展功能；

（6）控制器，根据指令译码信号实现控制信号的逻辑，还要实现ALU\_OP与OP、Func之间的逻辑关系；

（7）寄存器堆RegFile，提供32个MIPS通用寄存器；

（8）算术逻辑单元ALU，可以提供加或减的运算功能并输出运算结果。

（9）三个寄存器A、B、C，用于分别暂存RegFile的R1、R2输出和ALU的输出。

设计好多周期微程序控制器后，按照图2-2中的数据通路对各个部件进行连接即可。详细流程见实验步骤。

## 实验步骤

### 单周期硬布线MIPS CPU实现（指令译码、控制信号输出和ALU控制器逻辑）

首先完成单周期硬布线控制器的设计。前面已经列出了8条指令的操作码OP和功能码Func，因此在设计指令译码逻辑时，主要分为两步：

（1）首先根据OP是否为0将指令分为R型和非R型两类；

（2）①对于非R型指令，直接根据OP即可给出对应指令的译码信号；

②对于R型指令，则根据Func的取值确定指令的译码信号，同时使R\_TYPE标记输出为1。电路图如图2-3所示。



图2-3：指令译码逻辑模块电路图

下面根据指令译码信号实现控制器输出控制信号的逻辑。这里首先需要考虑各个部件在数据通路上的作用以及输入的来源，然后根据每个部件输入来源的数量以及条件确定相应控制信号的逻辑表达式。

（1）设计数据通路。根据图2-1以及需要实现的8条指令的功能特征，可以作出主要功能部件的输入来源，并由此设计出CPU的数据通路。

对于每种功能部件，再进一步统计其输入来源的种类数，若仅有1种输入来源，则在构建输出控制信号逻辑时直接将其连接到相应的输入上；若由多种输入来源，则需要引入多路选择器以确定信号的实际来源，如表2-4所示。

从表中可以看出，输入来源个数为1的部件端口有IM、R1#、R2#、ALU.A、S-EXT、DM.A、DM.Din，这些端口不需要设置选择器。

（2）列出剩余功能部件端口（PC、W#、Reg.Din、ALU.B）在不同指令下数据选择的条件以及各自产生的控制信号，如表2-5所示。对每个控制信号，将其产生该信号的条件进行逻辑或即可得到其逻辑表达式。

表2-4：CPU数据通路设计

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **编号** | **MIPS指令** | **PC** | **IM** | **RegFile** | | | | **ALU** | | **S-EXT** | **DM** | |
| **R1#** | **R2#** | **W#** | **Din** | **A** | **B** | **A** | **Din** |
| 1 | add | PC + 4 | PC | rs | rt | rd | ALU | RD1 | RD2 | 无 | 无 | 无 |
| 2 | slt | PC + 4 | PC | rs | rt | rd | ALU | 无 | 无 | 无 | 无 | 无 |
| 3 | addi | PC + 4 | PC | rs | 无 | rt | ALU | RD1 | S-EXT | i[15:0] | 无 | 无 |
| 4 | lw | PC + 4 | PC | rs | 无 | rt | DM.Dout | RD1 | S-EXT | i[15:0] | ALU | 无 |
| 5 | sw | PC + 4 | PC | rs | rt | 无 | 无 | RD1 | S-EXT | i[15:0] | ALU | RD2 |
| 6 | beq | PC + 4 + offset | PC | rs | rt | 无 | 无 | RD1 | RD2 | i[15:0] | 无 | 无 |
| 7 | bne | PC + 4 + offset | PC | rs | rt | 无 | 无 | RD1 | RD2 | i[15:0] | 无 | 无 |
| 8 | syscall | PC | PC | 无 | 无 | 无 | 无 | 无 | 无 | 无 | 无 | 无 |
| **来源种数** | | 2 | 1 | 1 | 1 | 2 | 2 | 1 | 2 | 1 | 1 | 1 |

表2-5：单周期硬布线控制器控制信号表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令/多路选择器控制信号** | **RegDst** | **RegWrite** | | **MemToReg** | **MemWrite** | **AluSrc** | **Beq** | **Bne** | **Halt** |
| add | 1 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| slt | 1 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| addi | 0 | | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| lw | 0 | | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| sw | 0 | | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| beq | 0 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| bne | 0 | | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| syscall | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

对表2-5中除Beq、Bne每一个选择信号为1的位置进行说明，列举如下：

①RegDst为1代表RegFile寄存器堆的W#（即要写入的地址）来源于rd字段，为0代表来源于rt字段；

②RegWrite为1代表将WD输入的数据写入寄存器堆的W#地址；

③MemToReg为1代表从存储器中输出数据到寄存器堆RegFile；

④MemWrite为1代表将ALU运算的结果写入存储器Memory；

⑤AluSrc为1表示ALU的第二个操作数来源于立即数扩展器S-EXT，为0表示来源于RegFile的R2输出；

⑥Halt为停机信号，为1时代表系统停机。

综上所述，根据指令译码信号实现控制器输出控制信号的逻辑电路图如图2-4所示。

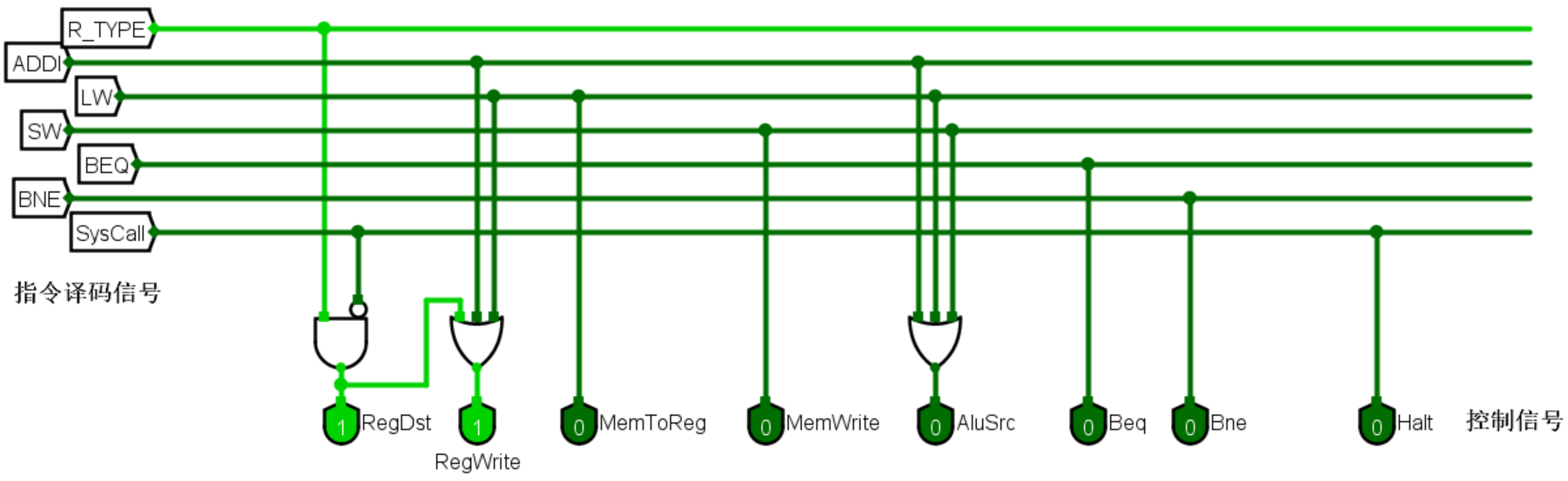


图2-4：根据指令译码信号实现控制器输出控制信号电路图

对于ALU控制器逻辑模块，首先ALU\_OP只可能有两种取值5和11，分别代表加法和减法运算。若Func=2a，说明要执行ADD操作，ALU\_OP取值为5，否则取值为11。因此ALU控制器的逻辑电路图如图2-5所示。

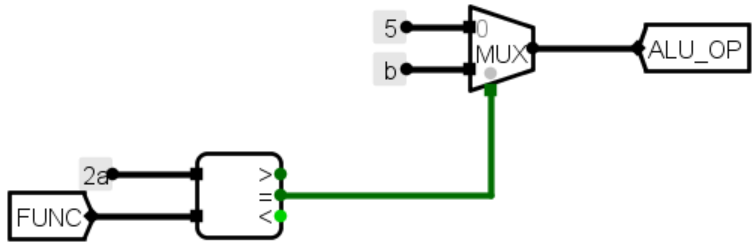


图2-5：ALU控制器逻辑（单周期硬布线）电路图

最后，按照图2-1连接单周期硬布线MIPS CPU电路即可。为了便于统计sort.hex程序执行的总周期数，将Halt指令取反后连接到一个计数器上，如图2-6所示。



图2-6：单周期硬布线MIPS CPU指令计数器电路图

### 多周期微程序MIPS CPU实现（ALU控制器逻辑、微程序地址转移逻辑）

现在要设计多周期的CPU，单周期和多周期之间存在的主要区别包括以下几点：

（1）多周期CPU仅包含一个存储器，相当于同时完成了单周期中指令存储器IR和数据存储器DR的功能；主要功能部件（如ALU、Mem）后都设有缓冲器，用于暂时保存输出的结果；

（2）一些功能部件（如ALU、RegFile）可在指令执行过程中的不同周期多次使用；

（3）指令计数器PC不再仅由时钟周期控制，这是因为不同指令的时钟周期数不同；

（4）ALU的输出由两种增加为三种，分别是输出指令bne与bne的分支地址、ALU的运算结果和指令lw与sw访问的存储器地址。由此需要对ALU控制器逻辑（即ALU\_OP的选择）做出修改，如图2-7所示。



图2-7：ALU控制器逻辑（多周期微程序）电路图

按照每条指令的功能（即表2-1），对照课上学习的指令数据通路详细流程，将取值阶段和8条指令对应的操作分解到多个机器周期内，如表2-6所示。

表2-6：取值阶段与各指令在不同机器周期内的数据通路

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指令** | **阶段** | **数据通路** | **状态** | **地址** |
| 取值阶段 | 取指令 | IR←(M[PC]), PC←(PC)+4 | S0 | 0 |
| 译码并取操作数 | A←(R[I[25:21]]), B←(R[I[20:16]]), C←(PC)+(S-EXT(I[15:0])<<2) | S1 | 1 |
| ADD | 加运算 | C←(A)+(B) | S2 | 2 |
| 写回 | R[I[15:11]]←(C) | S3 | 3 |
| SLT | 操作数比较 | C←((A)<(B)) | S2 | 2 |
| 写回 | R[I[15:11]]←(C) | S3 | 3 |
| ADDI | 加运算 | C←(A)+S-EXT(I[15:0]) | S4 | 4 |
| 写回 | R[I[20:16]]←(C) | S5 | 5 |
| LW | 计算地址 | C←(A)+S-EXT(I[15:0]) | S6 | 6 |
| 仿存 | DR←(M[PC]) | S7 | 7 |
| 写回 | R[I[20:16]]←(DR) | S8 | 8 |
| SW | 计算地址 | C←(A)+S-EXT(I[15:0]) | S9 | 9 |
| 仿存 | DR←(M[PC]) | S10 | 10 |
| BEQ | 送目标地址 | IF(A==B) PC←(C) | S11 | 11 |
| BNE | 送目标地址 | IF(A!=B) PC←(C) | S12 | 12 |
| SYSCALL | 停机 | -- | S13 | 13 |

以上每一行都是一个独立的微指令，对其进行状态编码为S0~S13，并补充标注在上表中。采用下址字段法并配合顺序控制位P（P为1表示控制存储器的地址由指令第一个阶段的微程序地址给出，P为0代表在执行阶段中下一条微指令地址由下址字段给出）。于是可以完成各指令不同阶段的控制信号，如图2-8所示。

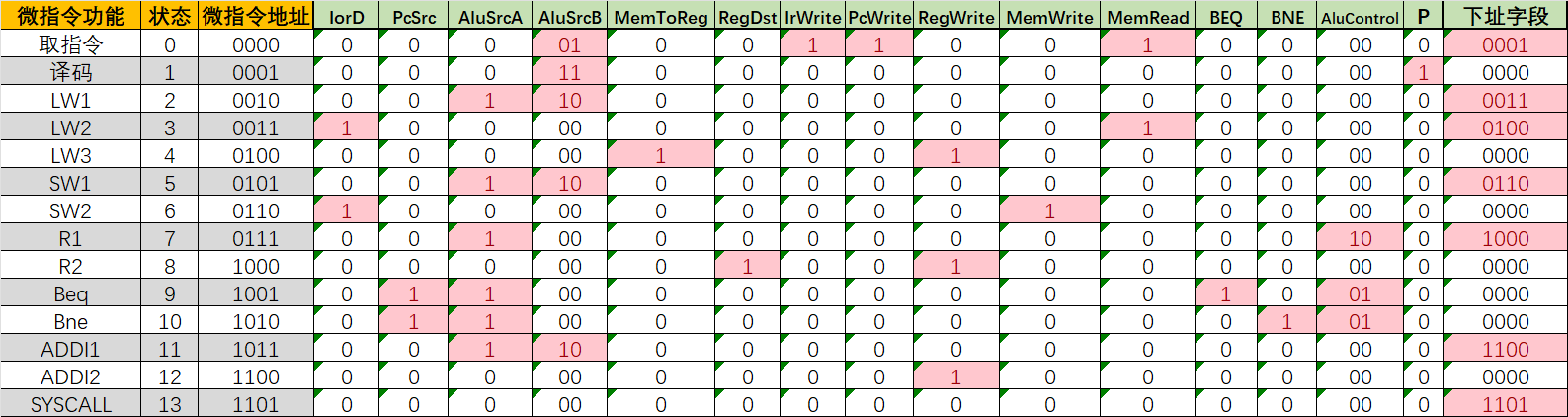


图2-8：多周期微程序MIPS CPU指令控制信号表

上图中各个控制信号说明如下：

（1）IorD：表示选择指令或数据，为1时代表取数据，为0时代表取指令；

（2）PcSrc：表示PC的输入来源，为1时代表跳跃寻址，为0时代表顺序寻址；

（3）AluSrcA：表示ALU的A端输入来源（PC或寄存器）；

（4）AluSrcB：表示ALU的B端输入来源（若为R型指令，则为寄存器输入；若为sw、lw、addu指令，则为立即数输入；若为跳转指令bne、beq，则为对应的偏移地址）；

（5）MemToReg：lw指令下写入寄存器的数据来源于存储器；

（6）RegDst：寄存器堆RegFile的W#输入端地址；

（7）IrWrite、PcWrite、RegWrite：指令寄存器、指令计数器、寄存器堆写使能端；

（8）MemWrite、MemRead：写/读内存控制信号；

（9）BEQ、BNE：BEQ、BNE指令译码信号；

（10）AluControl：ALU运算器控制信号，确定实现加法或减法。

根据上述状态和各状态之间的转移条件，进一步设计伪指令地址转移逻辑。依据图2-8给出的入口地址，可以确定微指令的地址转移逻辑如图2-9所示。



图2-9：多周期微程序MIPS CPU微指令地址转移逻辑

接下来，在Excel表格中“地址逻辑自动生成”下自动生成地址转移逻辑表达式。具体表达式为：，，，。

至此可以生成控制存储器所需的微指令并用16进制表示。在“微指令自动生成.xlsx”中填写每条微指令的下址字段和P后即可自动生成程序的微指令，如图2-10所示。

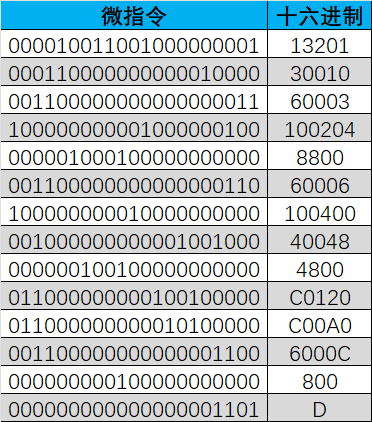


图2-10：多周期微程序MIPS CPU微指令编码

最后，按照图2-2连接多周期微程序MIPS CPU电路即可。为了便于统计sort.hex程序执行的总周期数，将当前微指令地址连接到一个计数器上，若微指令地址为d则说明是syscall指令，从而计数结束，如图2-11所示。

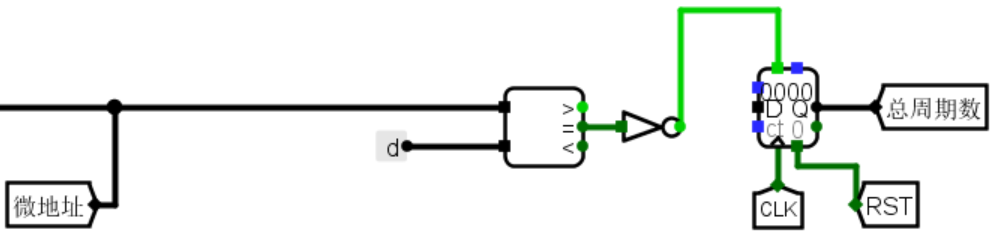


图2-11：单周期硬布线MIPS CPU指令计数器电路图

## 故障与调试

### 指令加载失败问题

**故障现象：**评测时一开始的IR为0，没有正确加载指令，如图2-12所示。

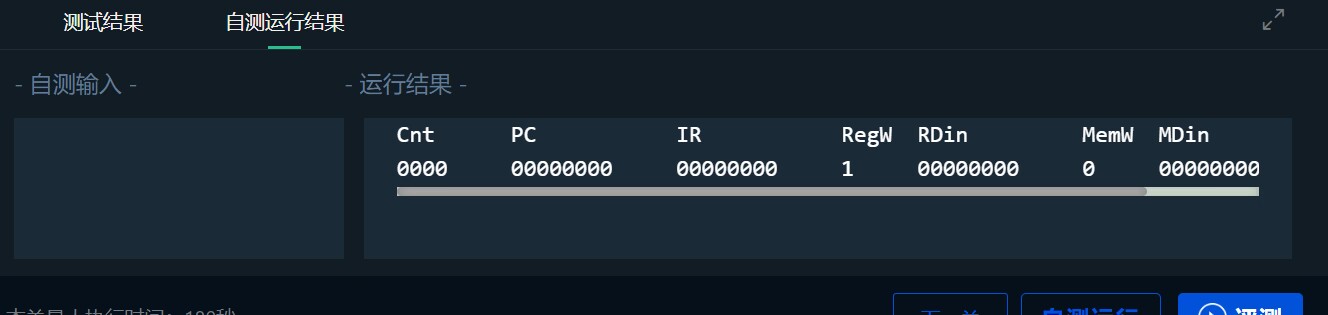


图2-12：指令加载失败

**原因分析：**在构建指令存储器时使用了RAM，导致本地测试正常，但线上评测环境不支持。

**解决方案：**将RAM（随机存储器）更换为ROM（只读存储器）即可。

### 数据地址不正确

**故障现象：**在单周期硬布线MIPS CPU下，冒泡排序sort.hex程序执行完成后，没有在预期的第80号存储单元形成降序排序的数据，而在第200号到第21c号存储单元之间可以观察到降序排序的数据，且相邻数据地址相差4。如图2-13所示。



图2-13：降序排序数据地址不正确

**原因分析：**对于数据存储器所用的RAM，只支持一次访问读出32位数据，直接给出字地址导致内存分布异常。

**解决方案：**将字地址除以4，相当于一次访问读出8位数据且相邻，即可解决这个问题。实验中的操作是使用分线器取出字地址的低6位，并将高2位置为0作为字节地址，然后送入RAM存储器的地址输入，就能在第80号存储单元处得到正确排序的数据结果，如图2-14所示。



图2-14：将地址修正后的降序排序数据

## 测试与分析

### 单周期硬布线MIPS CPU功能测试

首先向指令存储器中加载冒泡排序程序sort.hex，右键并点击“Load Image”，如图2-15所示。然后按Ctrl+K启动时钟，观察总周期数和程序执行完成后数据存储器中的内存情况。最终总周期数为224，与参考结果一致，如图2-16所示。数据存储器的内容如图2-14所示。

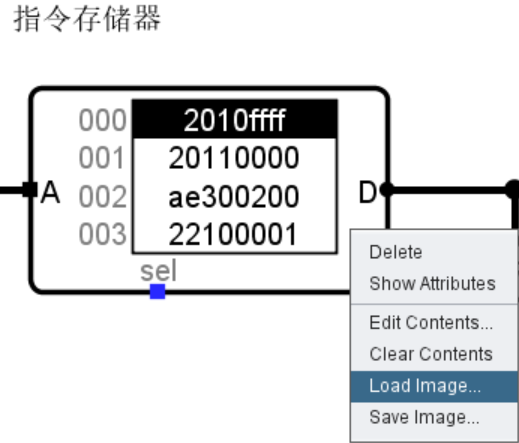


图2-15：加载冒泡排序程序

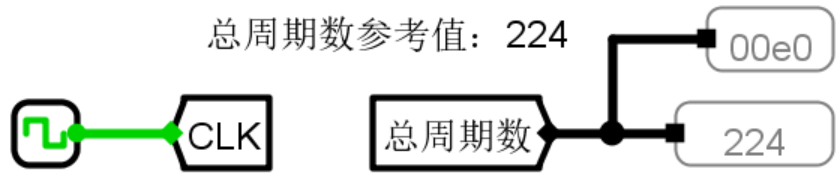


图2-16：单周期硬布线CPU冒泡排序程序总周期数

### 多周期微程序MIPS CPU功能测试

基本步骤与上面一致，这里还需要在控制存储器中载入提前准备好的微指令编码。最终总周期数为891，与参考结果一致，如图2-17所示。数据存储器的内容如图2-14所示。

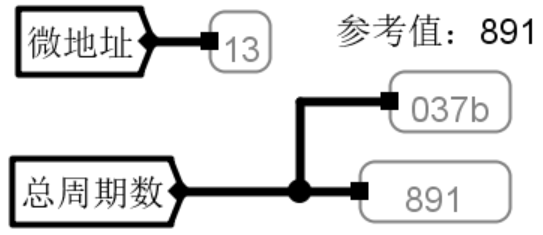


图2-17：多周期微程序CPU冒泡排序程序总周期数

## 实验总结

本次实验主要完成了如下几点工作：

1）完成了一个32位MIPS CPU处理器，包括单周期硬布线CPU和多周期微程序CPU，可以支持add、slt、addi、lw、sw、beq、bne、syscall共8条指令，并能运行给定的冒泡排序测试程序。

2）构建了完整的MIPS CPU数据通路，针对单周期和多周期的不同特点对数据通路的细节做出了完善。具体来说，实现了数据通路中寄存器读写、数据存储器读写、操作数运算、立即数拓展、控制信号的产生、32位指令的各部分分离与解析以及使用多路选择器选取正确的数据输入来源等功能。

3）实现了单周期硬布线控制器、多周期微程序控制器，包括指令译码信号、指令译码逻辑和ALU控制器逻辑；实现了对微程序指令的控制信号逻辑、状态编码，以及微程序的地址转移逻辑。

## 实验心得

1. 本次实验中，通过对单周期和多周期MIPS CPU的构建，我深入了解了计算机核心部件CPU的工作原理与运作机制，进一步掌握了MIPS指令集下R型、I型和J型指令的用法、区别与联系。同时，我也深入研究了数据通路的细节，使我对计算机系统的组成和运行过程有了更清晰的认识。
2. 在进行微程序指令编码时，我深刻感受到了微程序的灵活性与简便性。相较于硬布线控制器，微程序控制器的硬件设计简单，其中的微指令存储器和微指令控制逻辑也可以使用相对简单的存储器和逻辑电路来实现，具有良好的拓展性。这启示我在指令执行过程中可以根据实际需求采取多样化的方法。
3. 总体来说，做实验让我对课上讲到的知识有了一个实践的机会，很多课上的细节只有通过实验才能真正体会到，这也是实验课的一大作用。也希望后续有机会可以加强一下历次实验需求的具体描述和一些细节、常见错误等，使其看起来更清晰，也让同学们做实验时能更快上手。

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字: 李嘉鹏** |
| 二、对课程实验的学术评语（教师填写） |
|  |
| 三、对课程实验的评分（教师填写） |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 评分项目  （分值） | 课程目标1  工具应用  （10分） | 课程目标2  设计实现  （70分） | 课程目标3  验收与报告  （20分） | 最终评定  （100分） | | 得分 |  |  |  |  | |
| **指导教师签字:** |